

Abalone –Final Project Report

Benson Lee (bhl9), Hyun Joo Noh (hn57)

1. Introduction

This paper presents a minimax and a TD-learning agent for the board game Abalone. We had two goals in mind when we began our Abalone experiments, one for each of our agents.

For the TD-learning agent, we wanted to determine whether there was an issue with using several popular Abalone heuristics with TD-learning. The main problem we noticed with using published heuristics is that many of the heuristics use the symmetry of the board to assume that there is some sense of equivalence to all points equidistant from the board. The issue with this assumption is that, under the conventional board configuration, each player's pieces start out on one side of the board and tend to stick together for the majority of the game. A proper encoding should account for this by using a “subjective” board encoding with TD-learning; perhaps a player should treat its own side of the board differently from the opponent's side of the board. Given our time constraints, we do not look for an alternative board encoding; instead, we investigate this hypothesis by comparing agents trained on both symmetric and asymmetric initial board configurations. Towards this end, our results are inconclusive, but it seems that the initial board configuration does not have a significant effect on the quality of the agent. Luck seems to play a greater role; agents trained with the same parameters for the same amount of time can exhibit disparate performances.

The main issue with building a minimax agent is that the size of the branching factor prevents searching deeply into the game tree. ABA-PRO, a powerful minimax AI for Abalone, combats this problem by heuristically pruning off parts of the game tree that are unlikely to be reached. The problem with the heuristic used by ABA-PRO is that it is relatively expensive to evaluate. ABA-PRO finds the compactness of each player's pieces, which requires finding the distances between many marbles on a hexagonal board. Even after hashing the distances between each pair of tiles, we found that this computation requires a substantive amount of time when we ran our code on a profiler. In order to speed up our code, then, we used faster heuristics employed by other researchers. We wanted to determine whether this heuristic cutoff significantly affects the performance of the Abalone agent when we a different heuristic evaluation function to derive for game states. We find that the performance of the agent using heuristic alpha-beta is slightly worse than the performance of the agent that does not use heuristic alpha-beta pruning. This suggests that the heuristic pruning technique may be generally employed in searching the game tree of Abalone without great harm to the minimax approach.

2. Problem Definition and Algorithm

2.1 Brief Description of Rules:

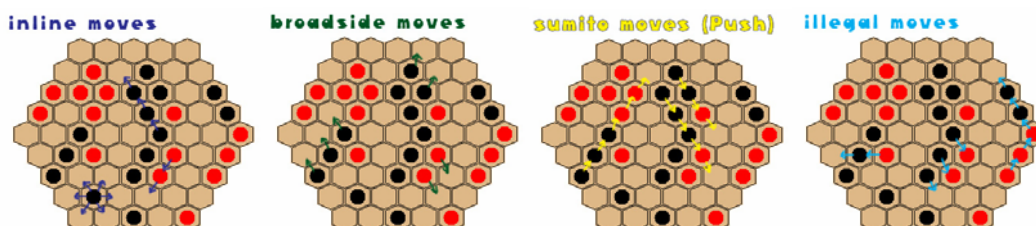


Figure 1: The images above describe legal and illegal moves. Broadside moves involve pieces moving sideways and inline moves involve pieces moving forwards.

Abalone is a popular strategy game that has sold millions of copies worldwide since its introduction a few decades ago. The rules are simple; the goal is to push 6 of the opponents pieces off the board. On any turn, a person may move 1, 2, or 3 pieces, if the pieces are adjacent and along the same axis. Each piece moves one space at a time as long as none of the destinations is occupied, and all pieces that a player selects must move in the same direction.

A player may push the opponent's pieces if the player's pieces outnumber the opponent's pieces in the direction of the move and the player's own pieces do not impede the push.

2.2 Abalone Complexity:

Game	Branching factor	log(state-space)	log(game-tree size)
Checkers	8-10	17-21	31
Othello	~5	28	58
Chess	30-40	46	123
Backgammon	~420	20	144
Xiangqi	75	75	150
Abalone	60-80	23	154
Go	360	160-172	360

Table 1: Complexity of different games, arranged in order of increasing game-tree size. The large branching factor for backgammon is misleading because it results from the probabilistic nature of the game.

One issue with creating an AI for Abalone is the complexity of the board game in comparison with other two-player, perfect information, zero-sum board games. The branching factor is estimated to be around 80, which is at least twice as high as chess, 8 times as high as Checkers, and 16x as high as Othello [1, 2]. The state space for Abalone is also many orders of magnitude greater than that of all of these games, making it very difficult to ensure that an agent thoroughly explores the state space (and that the input space is an adequate representation of the board game itself for TD-learning). The high game-tree complexity is a direct consequence of the branching factor of Abalone. Assuming a conservative branching factor of 60 for Abalone, the log-game-tree size has been estimated at 154 [6]. This figure is comparable to the log-game-tree size for Xiangqi, and exceeds that for many other board games.

Another issue is that the conventional definition of the game results in many stalemates, which cause games to be far longer than in other games. A typical human match takes a few hundred turns, and an AI match can take up to thousands of moves, even if the game is stopped after a state is revisited.

However, we are fortunate in that no move drastically changes the game board. This implies that it is feasible to produce a decent agent without resorting to deep search necessary for other games such as chess.

2.3 Our Algorithms:

As we mentioned above, we have implemented two different types of agents to tackle the Abalone board game.

Our standard minimax agent relies on a two-ply search of the game tree using score differences between the two players as the evaluation function of the game tree. In order to cut down on the number of tree nodes visited and speed up game play, alpha-beta pruning is applied to the result of our heuristic function. To optimize alpha-beta pruning, we presorted the tree nodes using a weighted linear combination of the following three heuristics:

Closeness to Center: sum of Manhattan distances of a player's pieces to the center of the board.

Number on Border: number of player's own pieces that border the edge of the game board.

Push: This variable is 1 if a push occurs in the move and -1 if a push does not occur in the move.

The TD-learning agent also uses the “Closeness to Center” heuristic and the “Number on Border” heuristic described above, but it does not use the push heuristic. In addition, the following heuristics for both players are also used as input:

Compactness: sum of the Manhattan distances between each pair of the player’s pieces

Number of Protected Pieces: A player’s piece is considered protected if all the adjacent board positions of that piece contain friendly pieces and that piece does not border the edge. In other words, this is the number of pieces that are surrounded by 6 other friendly pieces.

Threatened Pieces: A player’s piece is considered threatened if it is bordered by more enemy pieces than friendly pieces. This heuristic gives the number of threatened pieces for each player.

Number Near Center: number of pieces within 2 steps from the center tile

Number In Between: number of pieces not along the edge and not within 2 steps of center tile

The agent is based on Abalearn, an Abalone agent that uses TD-learning to achieve an intermediate level of play. The reward function used in the algorithm differs from that used in TD-Gammon in that a modulating parameter is added to force the Abalone agent to take some risks. The authors demonstrate that this improves the win rate of the agent, probably because fewer stalemates occur with a more aggressive agent. This risk parameter, originally analyzed by Mihatsch and Neuneier, involves multiplying the reward function by $1-\kappa$ if the reward function is greater than 0 and $1+\kappa$ if the reward function is less than 0 where κ takes a value between -1 and 1 exclusive. Intuitively, an agent with $\kappa > 0$ can be interpreted as a risk-avoiding agent because states promising higher immediate returns are overweighed, and agents with $\kappa < 0$ can be considered risk-seeking agents because states with lower immediate returns (but higher potential returns) are overweighed.

Our agent differed slightly from Abalearn in a few ways. We explicitly use a measure of compactness, something that Abalearn implicitly uses with the protected pieces and threatened pieces heuristics. Abalearn is initially trained on a random agent before it is trained on by self-play in order to learn a few characteristics of game play. By contrast, we chose to train solely by self-play, though we initialize ϵ to .1 before exponentially decreasing it 90% per step until ϵ is no higher than .01. The exponential decay used in Abalearn is slightly different, though not significantly different. We do this to prevent initial states from being overweighed in our TD-learning approach during training. In addition, we introduce a move penalty in addition to the risk parameter in order to speed up the game.

We also use a different stopping criterion. Abalearn maintains a database of previous board positions visited and stops the game once a board position has been revisited. This stopping point can be difficult to predict, and can, in certain cases, lead to extremely short games due to the shallow nature of our minimax search. In other cases, however, many more moves will be played using this stopping criterion. Due to time constraints, we choose to stop whenever either 200 moves has been made without a point being scored or a total of 1000 moves has been made. We informally observed the behavior of increasing these cutoffs (the former to 400, the latter to 2000), but found that there was no significant difference in the winning percentage of the agents, largely because these cutoffs are only hit after one of the agents has become comfortably conservative against the other. This restriction is not as stringent as it may seem. The average length of an Abalone game in one online play-by-email server was 87-ply, and an informal email in the MIT-Abalone mailing list describes a human tournament at the University of Waterloo in which even though each player was limited to a total 200 moves, the game never ended in a draw.

2.4. System Design

Our system consists of the following 6 classes:

Abalone:

-main class that processes command line arguments (save files and load files)

Board:

- stores information about board configurations and game parameters (i.e. score)
- contains convenience classes for referring to board

Move:

- contains move logic and determines lists of possible moves
- generates minimax moves and performs alpha-beta pruning
- generates random moves
- updates board state upon move and checks for end of game

Heuristic:

- contains all of the heuristics used by minimax and TD-learning, except for Push

TreeNode:

- inherits from Move class
- each instance is a node in the game tree representation of minimax
- processes minimax heuristic to recursively generate game tree

Backprop:

- inherits from Move class
- contains TD-learning algorithm
- saves and loads weight files used in TD-learning

3. Experimental Evaluation

3.1.1 TD-learning Agent - Methodology

For TD-learning, we first tested a self-play agent trained on 2000 games on a German Daisy board using default parameters to see if our TD-learning implementation functioned. We chose this configuration because it was symmetrical, unlike the conventional board configuration. We played it against a random agent 5000 games, and it was able to win 4919 of those games and reach a tie 18 times. For this test, we used the same parameters as those that performed best in the Abalearn paper, though our heuristics were slightly different so the two TD-learning agents are not exactly the same. Overall, the TD-learning agent won 29070 pieces against 1148 won by the random agent.

Next, we tuned the parameters of our agent by trying TD-agents trained on different parameters against a tuned minimax agent. These tests were performed to determine the optimal setting for the TD-learner using the German Daisy initial board configuration. The following parameters were tested:

Momentum (λ) – Parameter in neural net that determines influence from previous inputs

We used a default value of .7 and tried values of .1 and .35.

Learning rate (α) – Parameter that weights influence of previous rewards in neural net

We used a default value of .05 and tried .005 and .0005.

Discount factor (γ) – Parameter that weights influence of neural net state in RL

We used a default value of .5 and tried .1 and .7.

Risk (κ) – Risk parameter used to tweak risk-seeking/risk-aversion nature of TD-learner

We used a default of -.5 and tried setting this to -1, 0, and .5.

Move Penalty – Penalty for each move; prevents agent from being too conservative

We used a default of 0 and tried .001 and .0001.

Hidden Nodes – Number of hidden nodes in hidden node layer of neural net

We used a default value of 16 and tried 32 and 64 hidden nodes.

The Greek letters above refer to the formulas used in the Abalearn paper. The move penalty was something we introduced as an alternative to the Mihatsch and Neuneier risk parameter to encourage more aggressive play. We did not tune the epsilon parameter and its associated decay rate because the values of epsilon used in other TD agents we studied seemed to use similar values for epsilon.

For each of these parameters, we observed their effect on the TD-learner with 3 different values. We changed the values of 2 values at a time, training 70 agents overall, each with different parameter sets. For each of these agents, we trained them to self-play for 1000 games each. Using the weights found after 1000 training games, we then its performance against a tuned minimax opponent.

Using the TD-learner with the best set of parameters, we trained TD-agents on different initial board configurations. In this series of tests, we used the parameters found in the parameter tuning experiment above. We trained 2 TD-agents for on each of 5 different initial board configurations for 2000 games: the Conventional, German Daisy, Snakes, Alien Attack, and the Wall initial configurations. These agents were then tested against each other in the following manner: We randomly selected one of the agents to always plays black, and the other agent to always plays red. Games are played between every combination of agents, but the initial board configuration is always set as the board configuration on which the red agent was trained. In all, 500 games were played with each pair of agents.

The initial board configurations were chosen because they represent a diverse class of positions that could occur during game play. In the Wall and Conventional starting configurations, both players' pieces are biased towards one side even though they remain grouped together. As mentioned above, the German Daisy configuration is representative of a symmetric initial state. In the Snakes configuration, both players' pieces are again biased but are not clustered together. Both players' pieces are intermingled and ungrouped in the Alien Attack configuration. These different initial configurations are shown below.

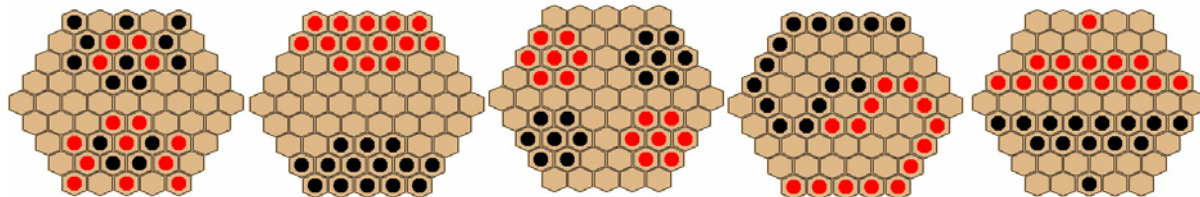


Figure 2: (from left to right) The Alien Attack, Conventional, German Daisy, Snakes, and Wall initial board configurations are shown above

We also played the agents that played as red against a minimax agent tuned for play on the German Daisy Board configuration to look at this problem from another angle.

3.1.2 TD-learning Agent - Results

Figure 3 shows the results of parameter tuning. For this experiment, we tested each TD-learning approach by having each TD-agent play 5 games against a minimax agent. Out of 70 parameter sets, only a small fraction performed as well or better than minimax. We found that 8 of the 9 parameter sets that resulted in a tie or a victory by TD used a learning rate of .005. Interestingly, the parameter set that performed best did not have a learning rate of .005. Its learning rate was set to .0005 and its move penalty was set to .0001. The learning rate seems to strongly influence the performance of our agent; in hindsight, this observation makes sense because the learning rate is used as a multiplier for the entire reinforcement learning value, which backprop uses to update the weights of the neural net.

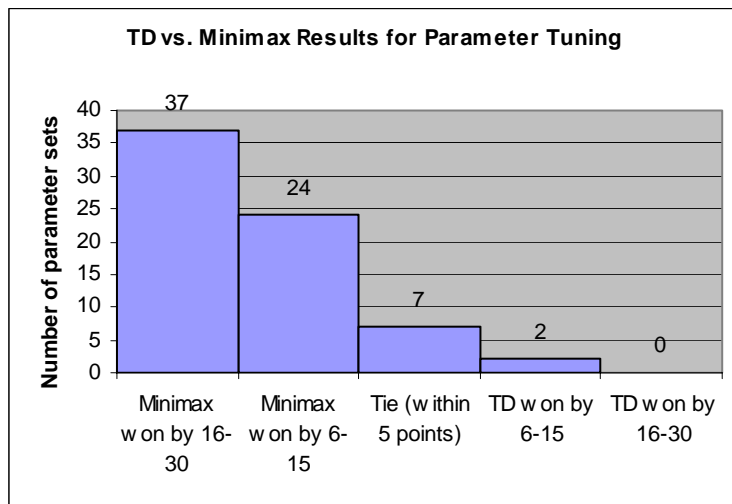


Figure 3: This chart shows the number of cases that resulted in score differences.

		Red color (trained on board played on)					Black's Record
Black color (starts first)		Alien Attack	Conventional	German Daisy	Wall	Snakes	
	Alien Attack	244-98 (1007-734)	500-0 (572-5)	412-0 (537-91)	415-1 (555-103)	226-108 (904-709)	0-5
	Conventional	131-148 (764-802)	11-102 (113-242)	500-0 (2999-48)	5-73 (66-165)	121-135 (630-677)	4-1
	German Daisy	171-199 (1196-1240)	34-41 (83-90)	37-2 (168-63)	0-0 (0-0)	173-196 (1203-1261)	3-1
	Wall	138-176 (889-960)	10-267 (124-542)	7-260 (98-487)	9-250 (129-518)	127-187 (879-1004)	5-0
	Snakes	137-59 (506-391)	67-4 (138-22)	62-1 (134-72)	60-1 (149-83)	151-58 (549-426)	0-5
	Red's Record (ex. diagonal)	2-2	2-2	3-1	2-1	1-3	

Table 2: Red wins-Black wins in 500 games, games always played on Red's board. The parenthesized numbers are the number of pieces that Red pushes off and Black pushes off respectively. Each square is colored the same as the superior agent for those tests. (Numbers do not add up to 500 because of draws).

The results in the table are mixed and inconclusive. The games along the diagonal from the upper left to the lower right of the table indicate tests in which both sides are trained with the same initial configuration that they are tested on. In four of these five sets of tests, one agent still clearly outperforms the other. With reservation, we tentatively believe that the initial board configuration does not seem to strongly influence the performance of the TD-learner. If the initial board configuration were very important, then the red agents would have outperformed the black agents by a greater amount. Luck seems to matter more; agents that perform well against one agent seem to perform well against the other agents. For example, the black agents trained on alien attack and snakes seem to be particularly weak, losing each of their games by a sizable margin. However, this observation cannot be generalized completely. The red German Daisy agent performs very well against four of the black agents, but does quite poorly against the black agent trained on the Wall configuration. In other words, the quality of the agents do not seem to exhibit a transitive property; if agent *A* is better than agent *B* and agent *B* is better than agent *C*, then agent *A* is not necessarily better than agent *C*. This characteristic certainly complicates our analysis of these results.

This set of agents did not perform nearly as well against the tuned minimax, even though they were trained using the same parameters that performed best against minimax during the parameter tuning phase of our experiments. What is interesting to note here is that even though the TD-learning agent trained on the German Daisy configuration never beats minimax, it performs best out of all of the TD-learning agents. This suggests that there may be a link between the board configuration and TD-learner performance. It is plausible that the parameters used for the German Daisy initial board configuration is suboptimal for use with TD-learners trained on different initial board configurations.

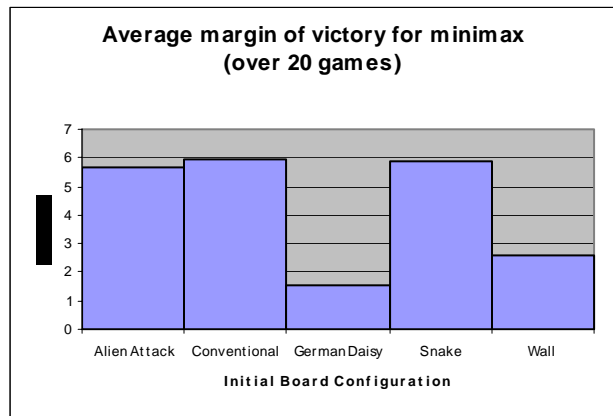


Figure 4: Minimax won soundly over TD from TD v TD experiments

3.2 Minimax Agent

We ran our agent against a random player in order to check whether the agent was working or not and to choose the best weights for each of the heuristics we used. For each of the 3 heuristics, we chose 6 different values of varying degrees (± 0.001 , ± 1 , ± 100) and ran 3 games per test set. With any positive value assigned to the push heuristic, whether it was 0.001 or 100, the minimax agent was able to defeat the random player all of the time. On average, the random agent scored .21 points per game on the minimax agent and the game ended in 212 moves regardless of the heuristic weight settings. This also helps confirm that our method of forcing a long game to end is not a very harsh bound.

As long as the weight for push heuristic was set to positive, there is no significant difference in the performance of minimax agent for different values of the weights. Based on this simple evaluation, it was not possible to tune the weights of the heuristic function more carefully. Since run time was an issue with the minimax agent, we simply chose the weight set that resulted in the minimum average number of moves made before winning the game, although this ‘average’ may not mean much since we only ran three games per test set.

After choosing the weights for the minimax heuristics, we went on to implement the heuristic alpha-beta pruning method used by ABA-PRO. In order to do so, we needed to determine which values of the heuristic function could be removed. We cannot set the cut-off as a fixed value because different parameter sets yield different heuristic values.

To solve this problem, we estimated the range of values that the heuristic function evaluates to by looking at different values observed during the first four moves by minimax. For these moves, we stored the heuristic values of all the nodes into an array. After the 4th move is made by the heuristic minimax player, we determined the ceiling and floor heuristic cut off values by sorting the heuristic value array and determining the 10th, 15th, 20th, 85th, and 90th percentile values of heuristic values. Recall that our minimax agent uses a 3-ply search of the game tree. In the first ply, all nodes with a heuristic value below the 10th percentile are removed. In the second ply, all nodes with a heuristic value below the 15th percentile and above the 90th percentile are removed. In the third ply, all nodes with a heuristic value below the 20th percentile and above the 85th percentile are removed.

To determine the quality of our heuristic minimax, we ran our heuristic minimax agent versus the original minimax agent using the 105 weight sets for which the minimax player performed significantly better than the random player from the previous test. We ran each of these tests twice. During the first set of runs, the heuristic minimax player starts the game. During the second set, the original minimax agent begins the game. We also observed the average number of nodes being observed by our agents in each step when playing against a random opponent to estimate the number of nodes discarded by heuristic alpha-beta pruning.

To see how much tree nodes are actually being discarded, we ran the original minimax agent with the parameters that resulted in the shortest average match against random for 100 games and observed the average number of nodes generated on each move. We repeated this experiment with the heuristic minimax agent with the same parameters in order to determine the approximate number of nodes we pruned using the heuristic cut-off method. The result is shown in Figure 5.

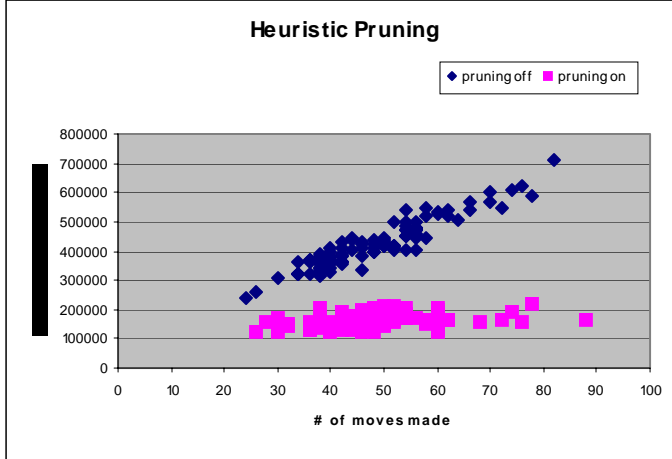


Figure 5: The number of nodes generated by heuristic alpha-beta pruning is constant in the number of nodes generated

able to push black on the following move. This splits up black's pieces, severely weakening the position of black. We suspect that this phenomenon may also be responsible for the dominance of the red agents in the TD vs. TD runs shown in table 2.

3.3 Discussion

Our results seem to indicate that our agents are mostly functional, though the tests we conducted with our agents remain inconclusive. We are hampered by the lack of data, which stems from the slow speed at which minimax runs. However, without minimax, it is difficult to conclusively determine the performance of any agent we use. The TD-agents lack the consistency to be a good measuring stick for other agents.

We ran our experiments on computers that would have been top-of-the-line when ABA-PRO was written, and their alpha-beta minimax (without heuristic pruning) ran at about the same speed as our alpha-beta minimax without heuristic pruning. What we ultimately need to be able to do to speed up our minimax is to prune a greater number of nodes. In order to do so, we have to find a heuristic that is more robust to this type of heuristic pruning, which would require us to further evaluate our minimax heuristic function. Since further evaluation necessitates more time devoted to this project, we have a chicken and the egg problem; heuristic alpha-beta pruning requires a good heuristic, which requires a good heuristic pruning function so that more tests can be run at a quicker pace. In other words, different heuristic functions may work better with different heuristic alpha-beta cutoffs, and it is difficult to account for both simultaneously. We sidestep this problem by using a somewhat arbitrary cutoff in which the cutoff increases by a constant amount for each additional ply that we search. Perhaps we would see improvements if we used a nonlinear function of the game-tree depth to determine our cutoffs. In addition, perhaps we would produce better estimates of our heuristic values if we occasionally updated the cutoffs based on the heuristic values generated in recent runs.

If we were to go on further in experimenting with the heuristic pruning and fine-tuning the cut off functions, we would experiment further with different criteria for deciding whether the heuristic cut off function discards right amount of nodes. The criterion used by ABA-PRO is that "playing on level $d+1$ with heuristic alpha-beta switched on should always be superior to playing on level d with the heuristic switched off" [1]. However, they do not seem to fine-tune the cutoff at each level very carefully. They always resolve this cutoff to the closest integer. As the tree grows deeper, there are increasingly few possible cutoffs since only the heuristic values close to 0 are considered. A better approach may be to consider deep searches into the game tree with a greater degree of detail than what is currently employed

As this graph shows, the heuristic pruning let us significantly reduce the number of tree nodes generated, but this was not enough to let us add an extra ply to the game tree. Overall, the performance of the heuristic pruning agent was slightly worse than the performance of the original minimax agent. When the heuristic agent was black, the average score was 5.3-2.7 in favor of the original agent. When the heuristic agent was red, the average score was 5.0-3.7 in favor of the heuristic agent. Interestingly, in both cases, the agent that was red won the game. We suspect that this is partially due to the German Daisy board configuration. If black moves towards red pieces at the beginning of the game, red is

by ABA-PRO. Deeper searches are more computationally expensive to tune, but the reward of more careful heuristic cutoff tuning may be a faster algorithm that can search slightly deeper into the game tree.

4. Related Works

We have alluded to most of the academic research done on Abalone in the sections above. ABA-PRO, ABLA, and Abalearn are the only agents we know of that were either developed by professional AI researchers. ABA-PRO is the most well-known, largely because it was the first Abalone AI to beat the world champion. Its success can be attributed to its ability to search much deeper than other agents into the game tree. The heuristic value it evaluates is relatively simple. It uses the compactness of each player's marbles and observes the distance between the center of mass and the center of the board for each player. The program is frequently used by Abalone players and was once distributed online, though they do not seem to distribute this program any longer. The authors also did not return our emails, so we were unable to try out their program.

There are an assortment of other agents available online, most of which seem to use a minimax algorithm that searches to a shallow depth. ABLA is representative of this approach. It uses even simpler heuristics than ABA-PRO (for efficiency) and search 3 and 4-ply deep into the game tree. Its performance was comparable to many other agents online.

Abalearn was the first TD-learning agent developed for Abalone. The agent differed from the TD-learning in that it incorporates a risk parameter and demonstrates that this risk parameter helps improve the performance of TD-learning. In fact, Abalearn claims to be the first application of the risk parameter described theoretically by Mihatsch and Neuneier. Overall, it was able to achieve an intermediate level of play.

One common problem with all of the agents we noticed is that all of them were evaluated on very few human players. Most of the agents online were not formally evaluated against human players. ABLA, for example, only considers performance against other agents that can be downloaded online. Abalearn and ABA-PRO were the only agents we found that analyzed performance against humans, but these evaluations were restricted in scope. Abalearn played against 3 human players on the official Abalone server, and the ABA-PRO paper only describes the agent playing against the Abalone world champion (1 person).

There are also a couple of unpublicized agents that are worth mentioning because they seem to be known by Abalone enthusiasts, even if there are no proper descriptions of the programs online. Nacre is interesting because it uses TD-learning to search more than 1-ply deep. According to its author, the value of searching 1-ply deeper is far greater than the benefit from using more board heuristics because there is not an obvious mapping between the game board and the neural net input (as with Backgammon). This author suggests that what is missing in ABA-PRO is that it does not know when its heuristic pruning approach becomes a disadvantage. In end-game situations, ABA-PRO may be pruning away moves that it could use to achieve victory in a limited number of steps. By knowing when to turn off heuristic pruning, the end-game behavior of ABA-PRO could be improved. For this, a pattern database of end-games such as that for chess AIs could improve the performance of Abalone agents.

My Lovely Abalone is a minimax agent created by a fan that, according to its author, has beaten ABA-PRO on the highest difficulty setting. There is very little information about this online, but the author does mention that he is trying to incorporate opening and end-game methods into his approach. Considering the dearth of recent papers on this board game, it seems plausible that the future theory about Abalone AI and programming will be driven by fan works rather than formal research papers.

5. Future Work

One thing we did not do in this experiment is to evaluate the performance of our agents based on games played against human players. Our shallow searches result in cycles occasionally, which may be exploited by a wily player that plays against the agent several times. Of course, none of the other agents performed extensive evaluations against human players, probably because of the time commitment this

requires. One large barrier is that there is no public server that an AI can easily connect to online; in fact, the authors of Abalearn manually inputted the predictions of their neural net in an online server in order to play against human players. This can be very time-consuming given the conservative nature of many AIs. In one case, they played a human agent for 3.5 hours before the game resulted in a tie. Thus, one way we could improve the efficiency of our testing (and the testing of other bots) is to host a server that people and AI agents could connect to play against each other.

Also, as mentioned in the evaluation of the minimax agent, the heuristic values of the game tree needs to be carefully analyzed, and the cutoff functions has to be more carefully chosen. We tried one heuristic cutoff function to prune the minimax tree, but perhaps tuning this function more cautiously could improve the speed of our implementation further. By improving speed, we may be able to search the game tree to a greater depth, which may also improve the performance of our implementation. With our current implementation, we do not excise enough nodes to be able to search significantly more deeply than we are currently searching.

For the TD-learning agent, one thing that can be done is tuning the learning rate of the agent more carefully since the learning rate seems to have a lot of influence on the overall performance of the TD-learning agent. In addition, we need to account for the erratic performance of training on different parameters by training several agents with each set of parameters. By doing so, we would have a greater understanding of which parameters are important to tune; we likely overlook the importance of certain parameters by not running additional tests. Of course, if our goal was solely to obtain the best performing agent, then this additional testing may or may not be worthwhile. It may simply be better to select whatever agent performs best against many other trained agents.

The TD-learning agent could also be improved by incorporating Q-learning. The current implementation of our TD agent considers only the state of the board configuration to evaluate its moves. The minimax agent relies on is the push heuristic by heavily weighting it relative to the static board heuristics that it uses. Thus, the performance of the minimax agent is heavily tied to the quality of this heuristic, so it is plausible that adding this heuristic to the TD-learning agent would significantly improve the performance of the TD-learner.

Lastly, one experiment we did not have time to conduct was testing the performance of minimax with heuristic pruning against the TD-learner. Since the TD-learners generally performed worse than the minimax agents, it is possible that it would be more meaningful to tune TD-learners on our heuristic minimax agents because it may be easier to distinguish parameter sets that are merely mediocre from the ones that are truly bad. Since the minimax agent currently wins so convincingly, all of the below average agents get lumped together into one indistinguishable group, preventing us from analyzing parameters that resulted in very bad TD-learning implementations.

6. Conclusion

Our results for the TD-learning agent are inconclusive, but we can tentatively claim that the initial board configuration does not seem to drastically affect the neural net. In a sense, this suggests that the rationale behind the inputs sent to the neural net is sound enough to be generalizable to various different board configurations. By making this statement, we do not mean to argue that the initial board configuration has no effect on the performance of the TD-learning agents. When TD-learners trained on different boards were run against minimax on the German Daisy initial board configuration, the TD-learner that performed best was the one that was trained on the TD-learner. We cannot make this claim convincingly because the performance of the TD-learner seems to vary greatly.

Our results for our minimax agent are easier to interpret than the results for our TD-learner. As with our TD-learning agent, it was able to soundly beat a random agent. It was also able to win against TD-learning agents in most cases, lending credibility to its strength. In addition, we demonstrated that heuristic pruning could significantly reduce the number of nodes we visit without detracting significantly from the strength of minimax. As anticipated, this suggests that the heuristic function that we evaluate on is smooth enough to prevent our heuristic minimax approach from being significantly hindered.

7. References

- [1] Aichholzer, O., Aurenhammer, F. and Werner, T. (2002). Algorithmic Fun: Abalone. *Special Issue on Foundations of Information Processing of TELEMATIK*.
- [2] Barto, A., and Sutton, R. (1998). Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA.
- [3] Campos, P., and Langlois T. (2003). Abalearn: Efficient Self-Play Learning of the game Abalone. INESC-ID, Neural Networks and Signal Processing Group, Lisbon, Portugal.
- [4] Ghory, I. (2004). Reinforcement Learning in Board Games. Department of Computer Science, University of Bristol.
- [5] Hulagu, B., and Ozcan, E. (2004). A Simple Intelligent Agent for Playing Abalone Game: ABLA. Proc. of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks, pp. 281-290.
- [6] Lemmens, N. (2005). Constructing an Abalone Game-Playing Agent. Bachelor Conference Knowledge Engineering, Universiteit Maastricht.
- [7] Mihatsch, O., and Neuneier, R. (2002). Risk-Sensitive Reinforcement Learning. Kluwer Academic Publishers, Hingham, MA, USA.
- [8] Norvig, P., and Russell, S. (2002). Artificial Intelligence: A Modern Approach (2nd Edition). Prentice Hall.
- [9] Persson, A. Using Temporal Difference Methods in Combination with Artificial Neural Networks to Solve Strategic Control Problems. KTH Numerical Analysis and Computer Science, Royal Institute of Technology, Stockholm, Sweden.